NRL Report 8840

Multiple Channel Adaptive Filtering Using a Fast Orthogonalization Network: An Application to Efficient Pulsed Doppler Radar Processing

KARL GERLACH

Airborne Radar Branch
Radar Division

September 28, 1984

NAVAL RESEARCH LABORATORY
Washington, D.C.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release, distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NRL Report 8840	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Research Laboratory 6b. OFFICE SYMBOL (*If applicable*)	7a. NAME OF MONITORING ORGANIZATION Naval Electronic Systems Command
6c. ADDRESS (*City, State, and ZIP Code*) Washington, DC 20375-5000	7b. ADDRESS (*City, State, and ZIP Code*) Washington, DC 20360
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Naval Electronic Systems Command 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*) Washington, DC 20360	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 62712N; PROJECT NO.; TASK NO. XF12-141-100; WORK UNIT ACCESSION NO. DN380-101

11. TITLE (*Include Security Classification*)
Multiple Channel Adaptive Filtering Using a Fast Orthogonalization Network: An Application to Efficient Pulsed Doppler Radar Processing

12. PERSONAL AUTHOR(S)
Gerlach, Karl

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Interim	FROM _____ TO _____	1984 September 28	22

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	Adaptive filtering; Radar; Gram-Schmidt decomposition; Doppler processing; Adaptive lattice

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

A numerically efficient algorithm is developed for adaptively filtering multiple input channels into desired multiple output channels. The algorithm is a type of adaptive lattice filter which employs a Fast Orthogonalization Network (FON) algorithm for numerical efficiency. Past researchers have concentrated on developing efficient lattice algorithms for the processing of stationary input channels; i.e., the input covariance matrix is Toeplitz in form. The algorithm developed in this report is designed to adaptively filter nonstationary input channels. Various implementations of the FON algorithm are given, and a performance measure based on the number of operations is formulated. An application of the technique to adaptive doppler processing is presented.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Karl Gerlach	22b. TELEPHONE (*Include Area Code*) (202) 767-3475 22c. OFFICE SYMBOL Code 5367

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

CONTENTS

MULTIPLE CHANNEL ADAPTIVE FILTERING USING A FAST ORTHOGONALIZATION NETWORK: AN APPLICATION TO EFFICIENT PULSED DOPPLER RADAR PROCESSING

I. INTRODUCTION

The direct adaptive filtering of multiple input channels by Gram-Schmidt orthogonalization has been the subject of intense research during the past decade [1-14]. The Gram-Schmidt technique (sometimes called the Adaptive Lattice Filter) has been shown to yield superior performance simultaneously in arithmetic efficiency, stability, and convergence times over other adaptive algorithms.

Arithmetic efficiency has been especially demonstrated in the filtering of stationary convariance sequences [4-14]. The stability of the algorithm is enhanced because it does not require the calculation of an inverse convariance matrix as does the Sample Matrix Inversion (SMI) algorithm of Reed, Mallet, and Brennan [15]. A overview of Adaptive Lattice Filters and a large bibliography on this subject are contained in Ref. 5.

In adaptive filtering, it is desirable to find the optimal weighting of multiple input channels such that the output signal to noise power ratio (S/N) is a maximum. The desired signal is associated with a desired signal column vector, $\mathbf{s}$, where $\mathbf{s} = (s_1, s_2, \ldots, s_N)^T$, N is the number of input channels, and T denotes the vector transpose. The vector component, $s_n, n=1, 2, \ldots, N$ represents the desired signal's component in the nth input channel. If $\mathbf{w}$ is an N-length column vector denoting the optimal weighting of the N input channels and $\mathbf{x}$ is an N-length column vector denoting the data from the N input channels, then it can be shown [16] that $\mathbf{w}$ must satisfy the following vector equation:

$$R_{xx}\mathbf{w} = \mu \mathbf{s}^*, \tag{1.1}$$

where

$$R_{xx} = E\{\mathbf{x}^*\mathbf{x}^T\}, \tag{1.2}$$

μ is an arbitrary constant which for convenience we set equal to one, $E\{\cdot\}$ denotes the expected value, and * denotes the complex conjugate. Equation (1.1) is often referred to as the Applebaum Adaptive Algorithm [16]. The matrix, R_{xx}, is called the input covariance matrix.

For some filtering applications, there may be as many output channels as there are input channels (such as a doppler processor). Hence, there will be N desired signal vectors. We define $\mathbf{S}$ to be the $N \times N$ steering matrix of desired signal vectors; i.e.,

$$\mathbf{S} = (\mathbf{s}_1\ \mathbf{s}_2 \ldots \mathbf{s}_N) \tag{1.3}$$

where $\mathbf{s}_n, n = 1, 2, \ldots, N$ are column vectors of the desirable signals. If $\mathbf{W}$ is defined as the optimal $N \times N$ weighting matrix, i.e., the weights that optimize the S/N in each of the output channels, then these weights satisfy the following matrix equation

$$R_{xx}\mathbf{W} = \mathbf{S}^*. \tag{1.4}$$

Problems occur in the solution for the weights if R_{xx} is ill conditioned. Due to computational inaccuracies, the algorithm can become unstable and the output channels extremely noisy. Adaptive lattice filtering does not normally exhibit stability problems.

Manuscript approved April 27, 1984.

In fact the weights formulated by Eq. (1.4) are not calculated at all when using adaptive lattice filtering. The data in the input channels are filtered directly through an orthogonalization network as is demonstrated in the following sections. However, the output channels will have the same or better (if R_{xx} is ill conditioned) S/N performance in each output channel as if the weights were calculated exactly in Eq. (1.4) and applied to the input data set.

Most of the research on Adaptive Lattice Filters has concentrated on the processing of stationary convariance sequences especially in the area of discrete-time linear prediction systems [4-14]. This implies that if r_{ij} is the i,j element to R_{xx}, then

$$r_{ij} = r_{i-j}. \tag{1.5}$$

Hence, the convariance matrix, R_{xx}, has the Toeplitz form.

In this report, we consider the efficient processing of channels that are not necessarily stationary with respect to one another so that Eq. (1.5) is not necessarily true. However, the input data on a given channel will be assumed stationary with respect to other data in that channel. The algorithm developed in the following sections will be a multichannel adaptive lattice filter which is structured for arithmetic efficiency in addition to retaining the good stability and fast convergence properties of orthogonalization networks. In Section VIII, we apply this algorithm (called a fast orthogonalization network) to implement an arithmetically efficient adaptive pulse doppler radar processor.

II. DECORRELATORS

Consider two channels of complex valued data: X_1 and X_2. We desire to form an output channel, Y, which is decorrelated with X_2; i.e.,

$$\overline{YX_2^*} = 0 \tag{2.1}$$

where the overbar denotes the expected value. This can be accomplished as follows. Let us write

$$Y = X_1 - wX_2 \tag{2.2}$$

and find a constant weight, w, such that Eq. (2.1) is satisfied. It can be shown that

$$w = \frac{\overline{X_1X_2^*}}{\overline{|X_2|^2}} \tag{2.3}$$

where $|\cdot|$ denotes the complex magnitude function and * denotes the complex conjugate. Figure 1 represents this decorrelation processor (DP).

In a digital implementation of the decorrelator, samples of the two input channels would be taken and the weight would be estimated. Let $\{X_i(1),X_i(2),\ldots,X_i(N_s)\}$ $i = 1,2$ denote the input data sequences for X_1 and Y_2 where N_s is the total number of discrete samples taken per channel. Then the decorrelation weight could be estimated as

$$w = \frac{\sum_{n=1}^{N_s} X_1(n)X_2^*(n)}{\sum_{n=1}^{N_s} |X_2(n)|^2}. \tag{2.4}$$

Note that Eq. (2.4) does not account for changes in the noise environment. If the noise environment is nonstationary, then such techniques as a "sliding window" or "forgetting factor" could be used on the input data. This is discussed further in Section V.

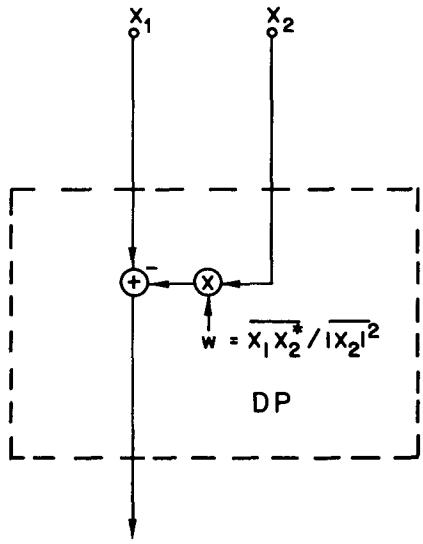

Fig. 1 — Decorrelation processor (DP)

Let us consider N channels of complex valued data: $X_1, X_2, \ldots, X_N$. To form an output channel, Y, which decorrelated with $X_2, X_3, \ldots, X_N$, we write

$$Y = X_1 - w_2X_2 - w_3X_3 - \ldots - w_NX_N. \tag{2.5}$$

We desire to find the weights, $w_n, n=2, 3, \ldots, N$ such that

$$\overline{YX_n^*} = 0 \quad n = 2, 3, \ldots, N.$$

We define a weight vector, $\mathbf{w} = (w_1, w_2, \ldots, w_N)^T$ where $w_1 \equiv 1$. It can be shown that $\mathbf{w}$ is the solution of the following vector equation:

$$R_{XX}\mathbf{w} = \mu \begin{pmatrix} 1 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{pmatrix} \tag{2.6}$$

where R_{XX} is the $N \times N$ convariance matrix of the input channels, i.e.,

$$R_{XX} = E\{\mathbf{X}^*\mathbf{X}^T\} \tag{2.7}$$

and $\mathbf{X} = (X_1, X_2, \ldots, X_N)^T$. The constant μ is not arbitrary but chosen so that $w_1 = 1$.

From Eq. (2.6), it is seen that the decorrelator could be implemented by taking data samples, forming a sample covariance matrix as implied by Eq. (2.7), solving Eq. (2.6) for the weights, and applying these weights to the input channels.

Another implementation of this decorrelation process is called Gram-Schmidt (GS) decomposition [1-5] as illustrated in Fig. 2 which uses the basic two-input DP as a building block [3]. GS decomposition decorrelates the inputs one at a time from the other inputs using the basic two-input DP as shown in Fig. 1. For example as seen in Fig. 2, in the first stage or level of decomposition, X_N is decorrelated with $X_1, X_2, \ldots, X_{N-1}$. Next, the output channel which results from decorrelating X_N with X_{N-1} is

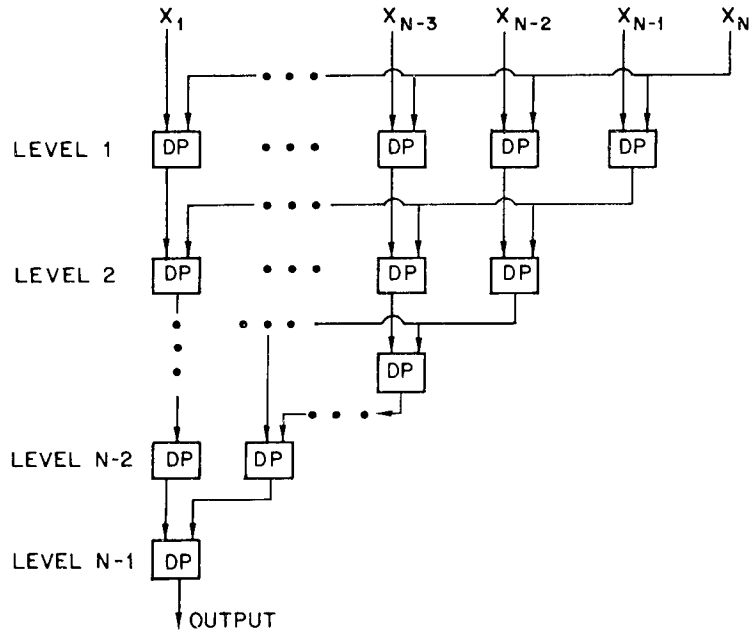

Fig. 2 — N-channel decorrelator

decorrelated with the other outputs of the first level of DPs. The decomposition proceeds as seen in Fig. 2 until a final output channel is generated. This output channel is totally decorrelated with the input: $X_2, X_3, \ldots, X_N$. Note that the GS decomposition is not unique; i.e., the order in which $X_2, X_3, \ldots, X_N$ are decorrelated from X_1 is arbitrary.

For N channels, the total number of DPs needed for GS decomposition is $0.5N(N-1)$. Hence, this number of decorrelation weights must be computed. For a digital implementation, these weights are determined sequentially. First, the first level weights are estimated after which the output data for the first level are calculated. These output data are used as inputs to the second level from which the second level weights can be calculated. The output data of the second level are generated by using these weights and the second level input data. The process continues until the $(N-1)$th level weight and outputs are calculated.

For notational purposes, we define the channel input appearing on the right-hand side of the DP as seen in Fig. 1 as being the input which is decorrelated with the channel appearing on the left-hand side. For the multiple channel case, all inputs appearing to the right of the far left input will be decorrelated from this input.

III. MULTIPLE CHANNEL ADAPTIVE LATTICE FILTER

In this section, we present a method for directly filtering multiple input channels into multiple output channels by using a multiple channel adaptive lattice filter and the concepts developed in the previous section.

Assume that the steering matrix as seen in Eq. (1.4), $\mathbf{S}$, is nonsingular. We configure a multichannel processor as seen in Fig. 3. The original input data column vector, $\mathbf{x}$, is multiplied by the matrix inverse of $\mathbf{S}^*$ to form another column vector, $\mathbf{X}$, which is also a multichannel process. Let $\mathbf{W}'$ be the optimal weighting matrix of $\mathbf{X}$ such that the S/N associated with each of the desired signals channels is maximized. It can be shown that $\mathbf{W}'$ satisfies the following matrix equation

$$R_{XX}\mathbf{W}' = I, \tag{3.1}$$

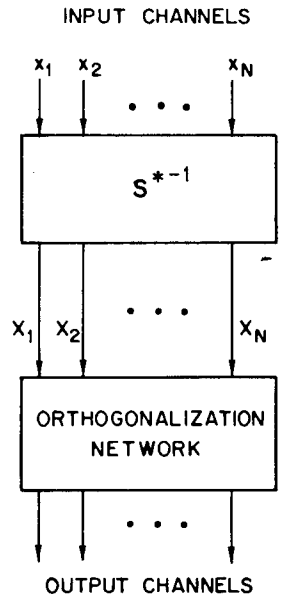

Fig. 3 — Multiple-channel adaptive lattice filter

where R_{XX} is defined by Eq. (2.6) and I is the $N \times N$ identity matrix. Actually

$$R_{XX} = \mathbf{S}^{-1} R_{xx} \mathbf{S}_t^{-1} \tag{3.2}$$

where t denotes conjugate transpose. The steering matrix, $\mathbf{S}^*$, has been transformed into a steering matrix which is the identity matrix.

If we examine the new desired signal vectors (which are the column vectors of the identity matrix), the nth channel has a desired signal vector:

$$(0\ 0 \ldots 0\ \underset{\uparrow}{1}\ 0\ 0 \ldots 0)^T$$

nth position.

Hence, it is seen that the form of Eq. (3.1) for the nth channel is very similar to Eq. (2.6) except that the "1" in the steering vector is not necessarily in thel first position as seen in Eq. (2.6). However, to perform the decorrelation process, it is only necessary to rearrange the input channels so that all other channels are decorrelated with the nth input channel, X_n, as seen in Fig. 4. By using this decorrelation procedure, the N channel orthogonalization network seen in Fig. 3 is now defined.

The ordering of the input channels for decorrelation as seen in Fig. 4 was arbitrary. It is shown in the next section that the input channels can be ordered so as to greatly reduce the required number of arithmetic operations. If there were no logic behind choosing the ordering of the input channels, it could be shown that the number of weights that are calculated by using this decorrelation procedure is $0.5N^2\ (N-1)$. In the following section we develop an algorithm which requires approximately $1.5N(N-1)$ weights for the same decorrelation process.

It is important to point out that the desired signals must be small or with a low duty cycle with respect to the noise in the respective channels. Otherwise, desired signals whose vector components are slightly different from the steering vector components will be cancelled due to the decorrelation process. Radar returns sampled in range are a good example of a low duty cycle desired signal where targets are sparsely distributed across the range bins.

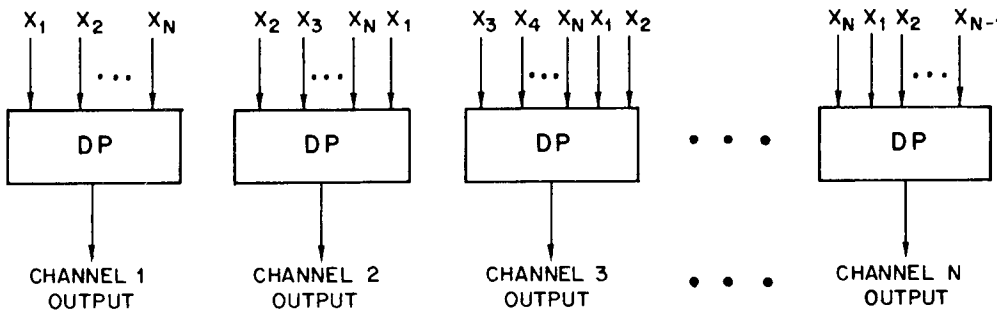

Fig. 4 — An arbitrary multichannel orthogonalization network

IV. FAST ORTHOGONALIZATION NETWORK

In this section, we present a methodology of configurating the two input decorrelation processors (DPs) to synthesize the N-channel orthogonalization network seen in Fig. 3 so that numerical efficiency is achieved. To this end, we introduce the following notation. A single channel decorrelator will be represented as

$$Ch_1 = [X_1, X_2, X_3, \ldots, X_N] \tag{4.1}$$

where $X_2, X_3, \ldots, X_N$ are decorrelated from X_1 and X_N is decorrelated first, X_{N-1} is decorrelated second, and so on. Figure 2 shows the structure of the correlator. The channel variable, Ch_1, references this structure to channel 1 or the X_1 channel. The X_n, $n = 1, 2, \ldots, N$ will be called the elements of the structure.

Numerical efficiency of the algorithm to be presented is achieved by taking advantage of redundancies that can occur for two different decorrelator structures. For example, let there be eight channels. Channels 1 and 4 can be generated as follows:

$$\begin{aligned} Ch_1 &= [X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8] \\ Ch_4 &= [X_4, X_3, X_2, X_1, X_5, X_6, X_7, X_8]. \end{aligned} \tag{4.2}$$

Note that Ch_1 and Ch_4 have the same four input channels at the far right. In the actual implementation, the substructure associated with these four rightmost channels can be shared by Ch_1 and Ch_4 as illustrated in Fig. 5. In fact, anytime two channels have exactly the same far-right channels as indicated by the decorrelator structure, the substructure associated with these far-right elements can be shared in the implementation process.

For convenience, let $N = 2^m$. In general we can configure 2^{m-1} output channels to have the common substructure of 2^{m-1} input channels; 2^{m-2} output channels to have the common substructure of 2^{m-2} input channels; and so on. Because of the structuring, the total number of weights that must be calculated will be approximately proportional to N^2. A further discussion of the number of arithmetic operations is given in the next section.

The following algorithm sequentially generates structures which can be implemented in a numerically efficient manner:

STEP 1 Generate root structure: $[X_1, X_2, \ldots, X_{2^m}]$.

STEP 2 Generate a structure which is the inverted order of root structure: $[X_{2^m}, \ldots, X_1]$.

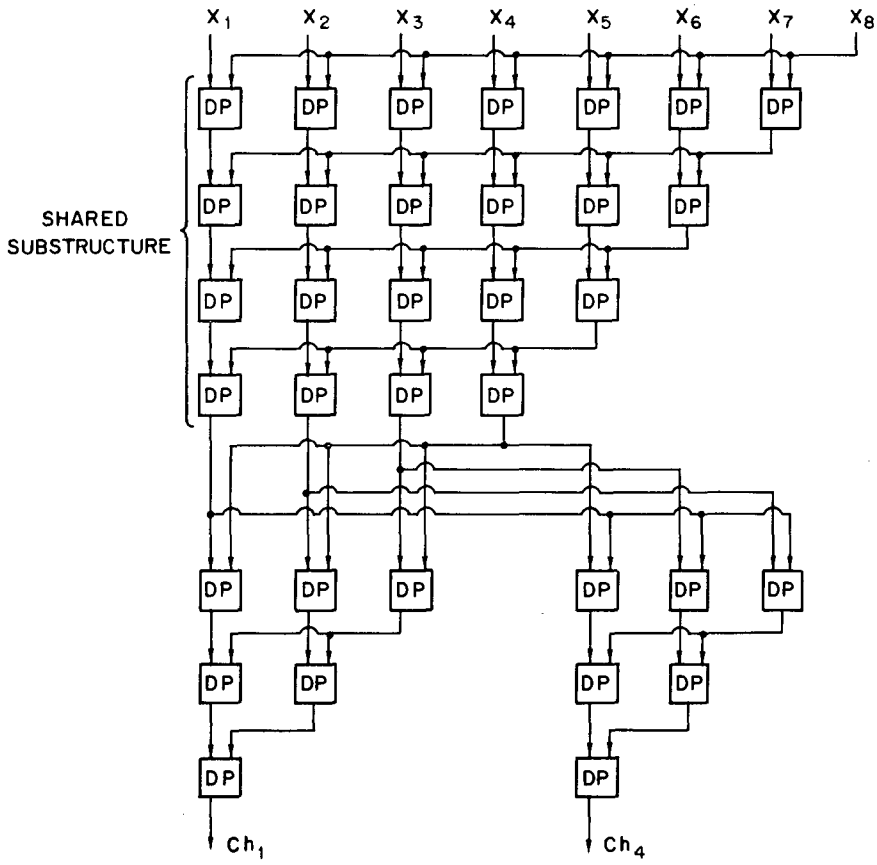

Fig. 5 — Example of substructure sharing

STEP 3 Generate 2 structures from the preceding 2 structures which have the first 2^{m-1} elements of the preceding structures in inverted order. All other elements remain the same.

STEP 4 Generate 4 structures from the preceding 4 structures which have the first 2^{m-2} elements of the preceding structures in inverted order. All other elements remain the same.

• • •
• • •
• • •

STEP k Generate 2^{k-2} structures from the preceding 2^{k-2} structures which have the first 2^{m-k+2} elements of the preceding structures in inverted order. All other element remain the same.

• • •
• • •
• • •

STEP $m+1$ Generate 2^{m-1} structures from the preceding 2^{m-1} structures which have the first 2 elements of the preceding structures in inverted order. All other elements remain the same.

For example, if $N = 2^3$ (where $m = 3$), the following structures would be generated sequentially by using the above procedure:

STEP 1 $[X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8] = Ch_1$

STEP 2 $[X_8, X_7, X_6, X_5, X_4, X_3, X_2, X_1] = Ch_8$

STEP 3 $[X_4, X_3, X_2, X_1, X_5, X_6, X_7, X_8] = Ch_4$

$[X_5, X_6, X_7, X_8, X_4, X_3, X_2, X_1] = Ch_5$

STEP 4 $[X_2, X_1, X_3, X_4, X_5, X_6, X_7, X_8] = Ch_2$

$[X_7, X_8, X_6, X_5, X_4, X_3, X_2, X_1] = Ch_7$

$[X_3, X_4, X_2, X_1, X_5, X_6, X_7, X_8] = Ch_3$

$[X_6, X_5, X_7, X_8, X_4, X_3, X_2, X_1] = Ch_6$

Note that channels 1, 2, 3, 4 have the substructure associated with X_5, X_6, X_7, X_8, and that channels 5, 6, 7, 8 have the substructure associated with X_4, X_3, X_2, X_1. Also note that Ch_1 and Ch_2 have the same 6-element substructure as do the channel pairs: (Ch_4, Ch_3), (Ch_8, Ch_7), and (Ch_5, Ch_6). A complete realization of the 8 output channels is illustrated in Fig. 6.

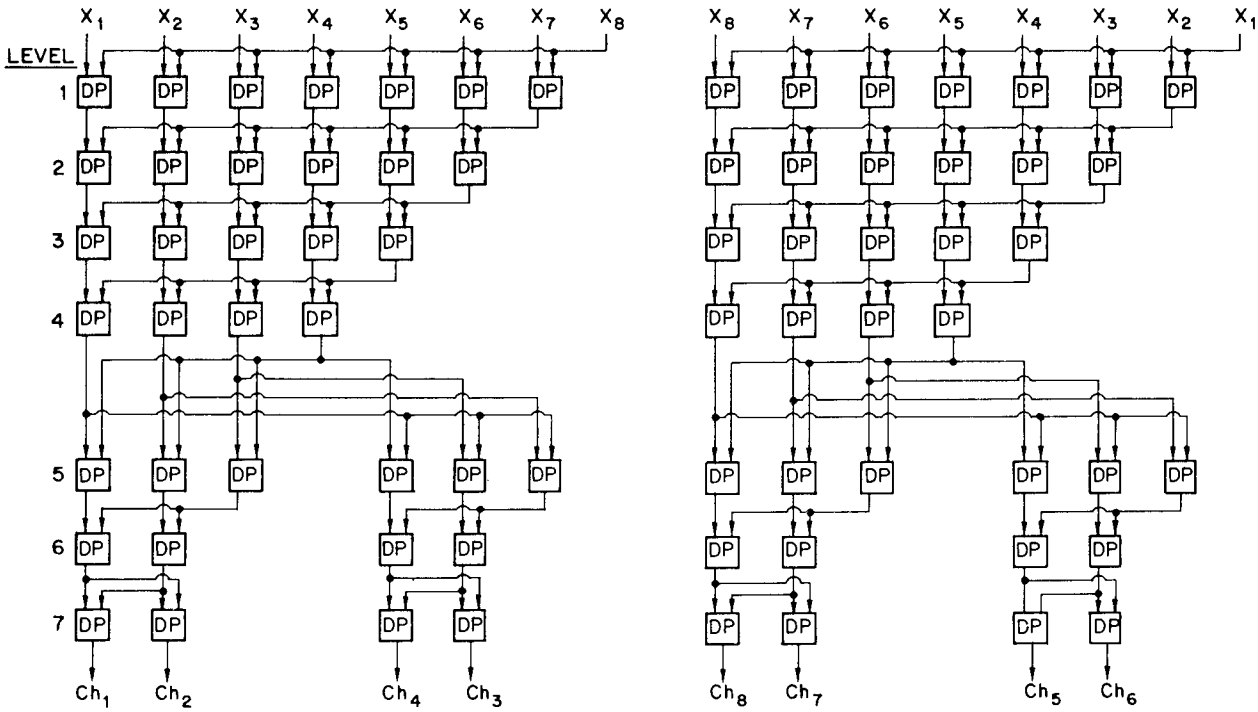

Fig. 6 — Complete realization of an eight-channel Fast Orthogonalization Network

V. NUMBER OF ARITHMETIC OPERATIONS

Each two input decorrelation processors (DPs) of the Fast Orthogonalization Network (FON) as depicted in Fig. 6 will have a complex weight associated with it. The number of DPs or complex weights associated with a FON can be found by considering the number of DPs at each level of the network. From Fig. 6, we see that the number of levels equals $N - 1$. If L_k is equal to the number of DPs at each level, then it can be shown that

$$
\begin{aligned}
L_1 &= 2^{m+1} - 2 \cdot 1 \\
L_2 &= 2^{m+1} - 2 \cdot 2 \\
L_3 &= 2^{m+1} - 2 \cdot 3 \\
&\vdots \\
L_{2^{m-1}} &= 2^{m+1} - 2 \cdot 2^{m-1} \\
L_{2^{m-1}+1} &= 2^{m+1} - 2^2 \cdot 1 \\
L_{2^{m-1}+2} &= 2^{m+1} - 2^2 \cdot 2 \\
&\vdots \\
L_{2^{m-1}+2^{m-2}} &= 2^{m+1} - 2^2 \cdot 2^{m-2} \\
&\vdots \\
L_{2^{m}-1} &= 2^{m+1} - 2^m \cdot 1.
\end{aligned}
\tag{5.1}
$$

Thus, the total number of DPs, N_{DP}, needed for a FON is derived by adding the right-hand sides of the above system of equations. It can be shown that

$$N_{DP} = \frac{3}{2} N(N-1) - \frac{1}{2} N \log_2 N. \tag{5.2}$$

The above number is also be equal to the total number of complex weights associated with a FON. The total number of operations associated with a FON is dependent on whether the algorithm is implemented (1) to recursively update the weights as new data samples arrive or (2) to calculate the weights as a function of a block of N_s data samples in each of the N channels.

Case I: Recursive Processing

Let $w(k)$ be a scalar weight associated with one of the two input DPs for the kth set of data samples ($k = 1, 2, \ldots, N_s$). Also let $u_R(k)$, $u_L(k)$, and $u_{out}(k)$ be the complex valued scalar right-side input, left-side input, and output of this particular DP respectively on the kth sample as seen in Fig. 7. In this block diagram, input $u_R(k)$ is decorrelated with input $u_L(k)$ and the decorrelated output is called $u_{out}(k)$. Also, we define two complex valued scalar state variables associated with this DP: $\nu_1(k)$ and $\nu_2(k)$. We see from Eq. (2.4) that $w(k)$ can be updated by using $u_R(k)$, $u_L(k)$, $\nu_1(k)$, and $\nu_2(k)$ as follows:

$$\nu_1(k) = (1-\alpha)\,\nu_1(k-1) + \alpha\, u_L(k) u_R{}^*(k) \tag{5.3a}$$

$$\nu_2(k) = (1-\alpha)\,\nu_2(k-1) + \alpha\, |u_R(k)|^2 \tag{5.3b}$$

$$w(k) = \frac{\nu_1(k)}{\nu_2(k)} \tag{5.3c}$$

where $0 < \alpha < 1$ is a constant which controls how fast past data are forgotten. This forgetting factor is necessary if the statistics of the input channels are time-varying.

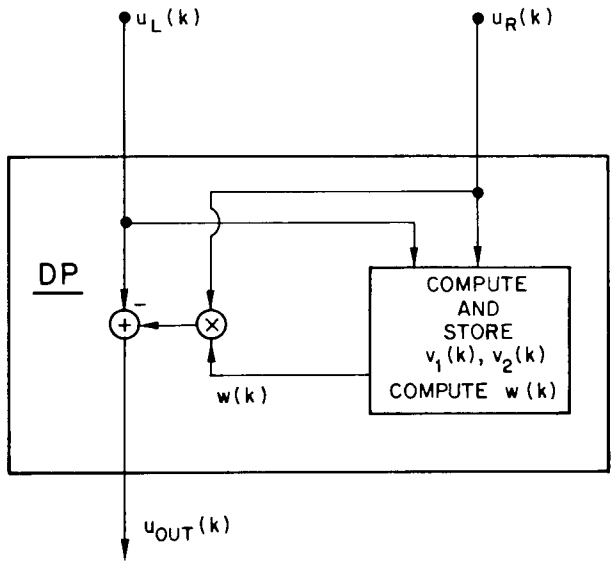

Fig. 7 — Recursive two-input decorrelator processor (DP)

The output of the specified DP has the form

$$u_{out}(k) = u_L(k) - w(k)u_R(k). \tag{5.4}$$

By inspecting Eqs. (5.3) and (5.4), we see that there are seven complex multiplication operations (CMOPs) and one complex division operation (CDOP) per DP per data step or iteration. If $N_{CMOP}^{(R)}$ and $N_{CDOP}^{(R)}$ are the number of CMOPs and CDOPs per iteration respectively, then by using Eq. (5.2), we can show

$$N_{CMOP}^{(R)} = 10.5N(N-1) - 3.5N \log_2 N \tag{5.5}$$

$$N_{CDOP}^{(R)} = 1.5N(N-1) - 0.5N \log_2 N. \tag{5.6}$$

Case II: Block Processing

For block processing, the total number of CMOPs, $N_{CMOP}^{(B)}$, is the sum of the number of CMOPs associated with finding the N_{DP} weights, and the number of CMOPs associated with processing the N_s input data samples per channel through a FON.

It can be shown by using Eq. (5.2), inspection of Eq. (2.4), and the fact that each DP must process N_s data points that

$$N_{CMOP}^{(B)} = N_s(4.5\ N(N-1) - 1.5\ N \log_2 N). \tag{5.7}$$

Since only one CDOP per DP is used in the block processing technique, it follows that

$$N_{CDOP}^{(B)} = 1.5\ N(N-1) - 0.5\ N \log_2 N. \tag{5.8}$$

Let us compare the arithmetic efficiency of the block-processed FON algorithm with the Sample Matrix Inversion (SMI) algorithm [15]. For the SMI there are $N_s N^2$ CMOPs needed to calculate the sample covariance matrix, R_{XX}, and approximately $N^3/3$ CMOPs required to find R_{XX}^{-1}. If the steering matrix is the identity matrix, then the weighting matrix equals R_{XX}^{-1}. Finally, there are $N^2 N_s$ CMOPs required to multiply the $N \times N$ weighting matrix times the $N \times N_s$ input data matrix. Thus, if $N_{CMOP}^{(SMI)}$ is the number of CMOPs needed to implement the SMI algorithm, then

$$N_{CMOP}^{(SMI)} \doteq \frac{1}{3} N^3 + 2N_s N^2. \tag{5.9}$$

It is shown in Ref. 15 that if the input channels are zero mean gaussian processes, then the average of the output S/N will be within 3 dB of the optimum S/N after $2N$ data samples per channel. Thus, if we set $N_s = 2N$, then for $N >> 1$

$$N_{CMOP}^{(B)} \doteq 9\ N^3 \tag{5.10}$$

and

$$N_{CMOP}^{(SMI)} \doteq 4.33\ N^3. \tag{5.11}$$

Hence we see that the FON algorithm is about half as fast as the SMI algorithm in attaining good S/Ns (within 3 dB of the optimum). However, we note that the FON algorithm does not require the inversion of a matrix which can lead to numerical instabilities if the sample covariance matrix is ill-conditioned.

VI. PARALLEL PROCESSING

From Fig. 6, it is seen that there are exactly $N - 1$ levels of DPs associated with a FON. From Eq. (5.1), we observe that the maximum number of DPs per level is $2N - 2$ and the minimum number is N. For the block processing algorithm described in the previous section we see that as the data (all $N_s L_{k-1}$ data points) are processed through the kth level that the input data may be discarded and the output data ($N_s L_k$ data points) become the new input data set. Hence $2N - 2$ parallel DPs could be configured as seen in Fig. 8.

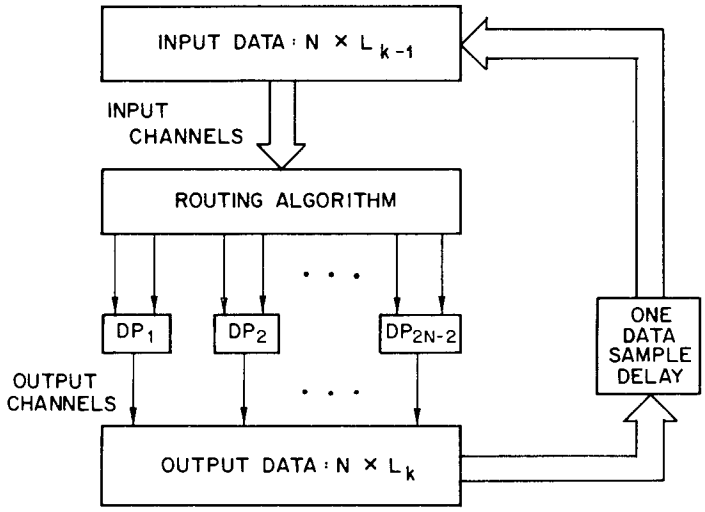

Fig. 8 — Parallel processing architecture of the Fast Orthogonalization Network (FON)

We allow these $2N - 2$ DPs to simultaneously perform all of the two-input DPs at a given level as shown in Fig. 6. However, for this to occur there must be a routing algorithm whose function it is to see that the L_{k-1} input data channels are inputted into the proper DPs. From Fig. 8, we see that the output data are stored back into the input data memory bank. After sequencing $N - 1$ times through the DPs, the algorithm is finished.

From inspecting Eq. (2.4), it is observed that the processing time through a DP is proportional to the number of data points per input channel, N_s. Since the bank of $2N - 2$ DPs is used $N - 1$ times, the total processing time, T_{PARA}, using a parallel architecture, is approximately proportional to the number of input channels and the number of data points per channel; that is,

$$T_{\text{PARA}} \sim NN_s. \tag{6.1}$$

Hence, parallel processing, by using the architecture seen in Fig. 8, can significantly decrease the processing time. This reduction occurs because of the inherent structure of the FON. In addition, the number of parallel DPs required is $2N - 2$ which reduces the hardware requirements.

VII. SOFTWARE ALGORITHM

A software algorithm called the Fast Orthogonalization Network (FON) algorithm has been devised which generates the 2^m decorrelator output channels from the 2^m input channels by using the common substructures of the various channels as described in Section IV (note that $N = 2^m$). Let each of the 2^m input channels have N_s sample points. Thus in the nth channel, $X_n(1), X_n(2), \ldots, X_n(N_s)$ are observed.

The algorithm requires at various points to reduce 2^k input channels to 2^{k-1} output channel through a partial orthogonization as illustrated in Fig. 9. The 2^{k-1} rightmost input channels are decorrelated with the 2^{k-1} leftmost input channels. If $U(I, J)$ are the input samples where $J = 1, 2, \ldots, 2^k$ indicates the channel number and $I = 1, 2, \ldots, N_s$ is the sample index, the following software algorithm called the kth Order Partial Orthogonalization algorithm generates the desired 2^{k-1} output channels:

1. Set $V^{(0)}(I, J) = U(I, J)$; $J = 1, 2, \ldots, 2^k$ $I = 1, 2, \ldots, N_s$
2. Set $h = 1$, $J_0 = 2^k$
3. Calculate recursively
$$V^{(h)}(I, J) = V^{(h-1)}(I, J) - V^{(h-1)}(I, J_0)\, W^{(h)}(J)$$
where
$$W^{(h)}(J) = \frac{\sum_{M=1}^{N_s} V^{(h-1)}(M,J)\, V^{(h-1)*}(M,J_0)}{\sum_{M=1}^{N_s} |V^{(h-1)}(M,J_0)|^2}$$
$$J = 1, 2, \ldots, 2^k - h; \; I = 1, 2, \ldots, N_s$$
4. Set $h = h + 1$, $J_0 = J_0 - 1$
5. If $h \leqslant 2^{k-1}$ GO TO 3
6. end

The final outputs are contained in $V^{(2^{k-1})}(I,J)$.

The FON algorithm requires that the input channels, $X_1, X_2, \ldots, X_N$, be commutated so that the final output channels of the FON are properly aligned. (This problem also occurs with the FFT algorithm, but a commutation algorithm matches the proper output channel with the input channel.) By properly aligned, we mean that the kth output channel of the FON algorithm is decorrelated with input channels: $1, 2, \ldots, k-1, k+1, \ldots, N$. The following algorithm called the Commutated Indexing algorithm computes the commutated indices and stores them in an N element array called INDEX.

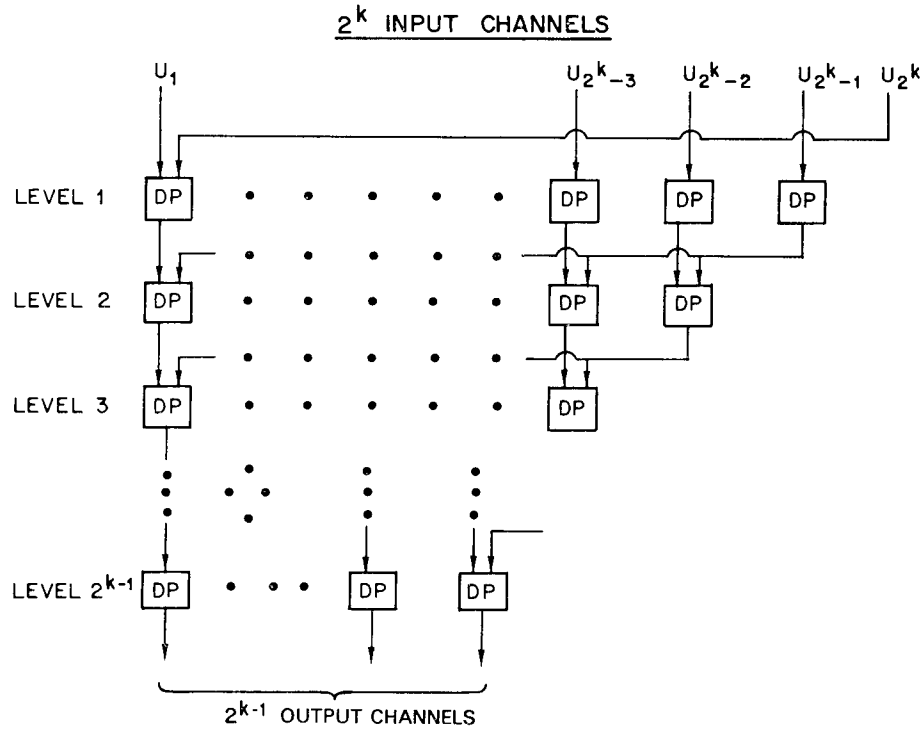

Fig. 9 — Partial orthogonalization, 2^k input channels to 2^{k-1} output channels

1. Set INDEX (1) = 1, INDEX (2) = 2
2. Set $J = 1$
3. Set INDEX $(2^J + k)$ = INDEX $(k) + 3 \cdot 2^{J-1}$; $k = 1, \ldots, 2^{J-1}$
4. Set INDEX $(2^J + k)$ = INDEX $(k) + 2^{J-1}$; $k = 2^{J-1} + 1, \ldots, 2^J$
5. $J = J + 1$
6. If $J < m$ GO TO 3
7. end

After defining these preliminary algorithms, we now give the complete FON algorithm:

1. Input $X(I, J)$ array; $J = 1, 2, \ldots, 2^m$, $I = 1, 2, \ldots, N_s$
2. Calculate the commutated indices by using the Commutated Indexing algorithm.
3. Transfer $X(I, \text{INDEX}(J)) \rightarrow Y(I, J)$

   $J = 1, 2, \ldots, 2^m$, $I = 1, 2, \ldots, N_s$
4. Set $k = m$
5. Set $L = 0$

6. Transfer $Y(I, J + 2^kL) \rightarrow U(I, J)$

   $J = 1, 2, \ldots, 2^k$; $I = 1, 2, \ldots, N_s$

7. Calculate $V^{(2^{k-1})}(I, J)$; $J = 1, 2, \ldots, 2^{k-1}$, $I = 1, 2, \ldots, N_s$ by using the kth Order Partial Orthogonalization algorithm.

8. Transfer $V^{(2^{k-1})}(I, J) \rightarrow T(I, J)$

   $J = 1, 2, \ldots, 2^{k-1}$, $I = 1, 2, \ldots, N_s$

9. Transfer $Y(I, 2^kL + J) \rightarrow U(I, 2^k + 1 - J)$

   $J = 1, 2, \ldots, 2^k$; $I = 1, 2, \ldots, N_s$

10. Calculate $V^{(2^{k-1})}(I, J)$; $J = 1, 2, \ldots, 2^{k-1}$; $I = 1, 2, \ldots, N_s$ by using the kth Order Partial Orthogonalization algorithm.

11. Transfer $T(I, J) \rightarrow Y(I, 2^kL + J)$

    $J = 1, 2, \ldots, 2^{k-1}$, $I = 1, 2, \ldots, N_s$

12. Transfer $V^{2^{k-1}}(I, J) \rightarrow Y(I, 2^kL + 2^{k-1} + I)$

    $J = 1, 2, \ldots, 2^k$; $I = 1, 2, \ldots, N_s$

13. $L = L + 1$

14. If $L \leqslant 2^{m-k}$ GO TO 6

15. $k = k - 1$

16. If $k > 0$ GO TO 5

17. end

VIII. AN APPLICATION: ADAPTIVE DOPPLER PROCESSING

Adaptive filtering can be applied to radar doppler filter design [17,18]. Doppler filters are designed to accept doppler frequency shifted moving targets while rejecting the returns from the target background (clutter). The clutter is usually slow moving so that its energy is normally concentrated about the zero doppler frequency. A bank of filters is used to cover the entire doppler band; i.e., the doppler band is equally divided into subbands. Ideally, it would be desirable to place a rectangular bandpass filter about each subband so that the large clutter return is completely rejected out of band. However, only approximations of this rectangular filter are realizable. It has been shown [17,18] that adaptive doppler processing yields superior signal-to-clutter power ratio improvement performance over these approximate rectangular filter implementations. This results because each doppler filter is designed not only to accept a desired signal but also to place nulls at frequencies out of band where clutter returns exist. Each doppler filter is optimized with respect to the doppler filter's allocated subband by use of the Applebaum algorithm.

The input channels, x_n, $n = 1, 2, \ldots, N$, are formed by taking time-delayed samples (usually one pulse repetition interval (PRI)). Hence if $r(t)$ is the received radar signal, then

$$x_n = r(t - nT), \quad n = 1, 2, \ldots, N. \tag{8.1}$$

There are N weights associated with each of the N doppler filters. These N^2 weights, $\mathbf{W}$, are the solution of the following matrix equation:

$$R_{xx}\mathbf{W} = \mathbf{S}^* \tag{8.2}$$

where R_{xx} is the covariance matrix of the N input channels. If the input data are statistically stationary in time, then R_{xx} is a Toeplitz matrix. The matrix $\mathbf{S}$ is the matrix of steering vectors signifying the various doppler filter subbands. In general, the steering matrix has the following form:

$$\mathbf{S} = \begin{bmatrix} a_1 & a_1 & a_1 & \cdots & a_1 \\ a_2 & a_2\Gamma_N & a_2\Gamma_N^2 & \cdots & a_2\Gamma_N^{2(N-1)} \\ a_3 & a_3\Gamma_N^2 & a_3\Gamma_N^{(2)(2)} & \cdots & a_3\Gamma_N^{2(N-1)} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_N & a_N\Gamma_N^{N-1} & a_N\Gamma_N^{2(N-1)} & \cdots & a_N\Gamma_N^{(N-1)(N-1)} \end{bmatrix} \tag{8.3}$$

where $\Gamma_N = \exp\{-j\,2\pi/N\}$ and $j = \sqrt{-1}$. For the Brennan and Reed doppler processing algorithm [17], $a_n = 1$, $n = 1, 2, \ldots, N$. For this algorithm, each of the N doppler filters is optimized at one particular doppler frequency by use of the Applebaum algorithm. The particular doppler frequency is chosen to be at the center of the given subband. However, because in general the doppler shift is unknown within a given subband, any desired signal whose doppler is not at the center of one of these subbands will not be properly matched and signal detection will degrade especially at dopplers that are close in to the clutter spectrum.

Andrews [18] devised a steering matrix that gives superior performance. For this algorithm the doppler shift is assumed unknown across a particular subband, and the filter response is optimized over the entire subband. In essence, the best N-point finite impulse response (FIR) filter is fitted to a desired rectangular filter centered in a particular subband with a phase bandwidth of $2\pi/N$. Andrews shows that these weights are given by

$$a_n = \frac{\sin\left[\frac{\pi}{N}\left[n - \frac{N+1}{2}\right]\right]}{\frac{\pi}{N}\left[n - \frac{N+1}{2}\right]}, \quad n = 1, 2, \ldots, N. \tag{8.4}$$

The form of $\mathbf{S}^*$ as given by Eq. (8.3) is such that it can be factored as

$$\mathbf{S}^* = \mathbf{AB} \tag{8.5}$$

where $\mathbf{A}$ is a diagonal matrix with diagonal elements, $a_n{}^*$, $n = 1, 2, \ldots N$ and

$$\mathbf{B} = (\Gamma_N^{-(n-1)(l-1)}); \quad n, l = 1, 2, \ldots, N. \tag{8.6}$$

If we employ the multiple channel adaptive lattice filter by using a FON as depicted in Fig. 3, we see that the N input data channels, $x_1, x_2, \ldots, x_N$, are transformed by $\mathbf{S}^{*-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$. Due to the special form of the matrix, $\mathbf{B}$, it can be shown that

$$\mathbf{B}^{-1} = \mathbf{B}^{*}. \tag{8.7}$$

Therefore, the N output channels, $\mathbf{X} = (X_1, X_2, \ldots, X_N)^T$, that result by multiplying the N input channels by $\mathbf{S}^{*-1}$ are given by

$$\mathbf{X} = \mathbf{B}^{*}\mathbf{A}^{-1}\mathbf{x}. \tag{8.8}$$

Equation (8.8) indicates that the input data channels, $\mathbf{x}$, are first weighted by the inverse of the diagonal matrix $\mathbf{A}$, which is equivalent to weighting the nth channel by $1/a_n$, $n = 1, 2, \ldots, N$. The weighted channels are then multiplied by the matrix, $\mathbf{B}^{*}$. It can be shown that the transformation that results by multiplying a set of N channels by $\mathbf{B}^{*}$ can be implemented by using a fast Fourier transform (FFT) if $N = 2^m$. The covariance matrix, R_{xx}, is Toeplitz. However, note that the output channels of the FFT are not stationary with respect to each other; i.e., R_{XX} is not a Toeplitz matrix. The N outputs of the FFT are then processed by using the Fast Orthogonalization Network discussed in the previous sections. Figure 10 is a simplified diagram of this processor.

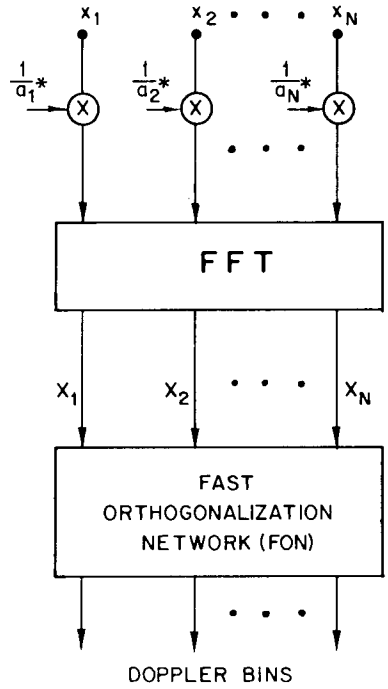

Fig. 10 — Adaptive doppler processor using a lattice filter

Let the input data set have the form depicted in Fig. 11. Here the input data are arranged so that the returns from sequential range cells are sequential samples of a given channel and the data in each channel at a given range are the time-delayed (one PRI) return of the preceding channel. In this figure, there are N_s range bins with R_0 the minimum range considered and ΔR is the range resolution. Returns from a given range bin occur one per PRI time step. If N PRIs are in the processing time window, then the returns in the nth PRI form the nth input channel; i.e., r_{mn} is the radar return from the mth range cell in the nth PRI interval. If this input data set is block processed by using the adaptive doppler processor as seen in Fig. 10, then the output data set will have the form depicted in Fig. 11. This matrix of output data will have elements, r'_{nm}, corresponding to the returns in a given range-doppler bin. If blocks of input data are sequentially processed, then the resultant output data sets can be inputted into a postdetection processor for the detection and tracking of targets.

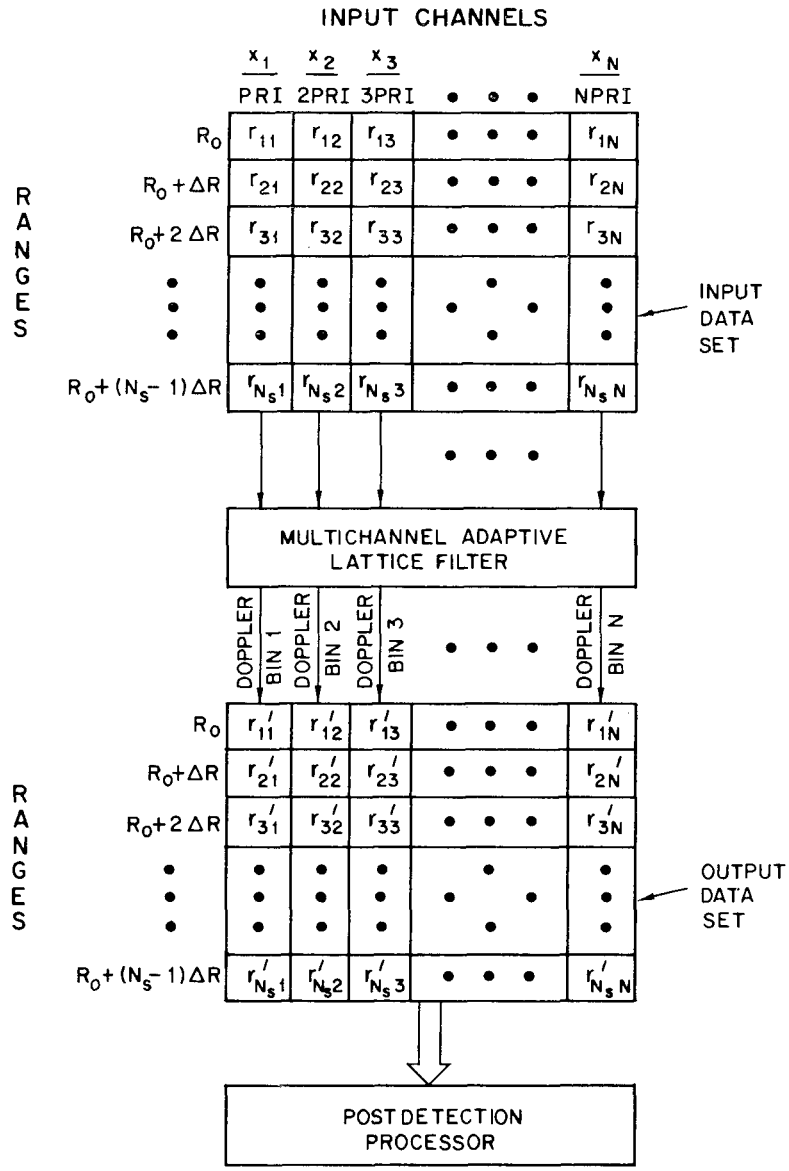

Fig. 11 — Input and output data sets for adaptive doppler processing

IX. CONCLUSIONS

A numerically efficient algorithm has been developed for adaptively filtering multiple input channels into desired multiple output channels. The algorithm is a type of adaptive lattice filter which employs a Fast Orthogonalization Network (FON) algorithm for numerical efficiency. Past researchers have concentrated on developing efficient lattice algorithms for the processing of stationary input channels. The algorithm developed in this report was designed to adaptively filter nonstationary input channels. Various implementations of the FON algorithm were given, and a performance measure based on the number of operations was formulated. An application of the technique to adaptive doppler processing was presented.

X. REFERENCES

1. B.L. Lewis, F.F. Kretschmer Jr., unpublished work of limited distribution, dating back to February 1974.

2. R.A. Monzingo and T.W. Miller, *Introduction to Adaptive Arrays*, Chapter 8, John Wiley and Sons, 1980.

3. W.F. Gabriel, "Building Block for an Orthonormal-Lattice-Filter Adaptive Network," NRL Report 8409, July 10, 1980.

4. M.A. Alam, "Orthonormal Lattice Filter—A Multistage, Multichannel Estimation Technique," *Geophysics* **43**, 1368-1383, Dec. 1978.

5. B. Friedlander, "Lattice Filters for Adaptive Processing," *Proc. IEEE* **70**(8), 829-867, Aug. 1982.

6. J.M. Delosme, "Algorithms and Implementations for Linear Least-Squares Estimation," Ph.D. dissertation, Stanford University, Stanford, Calif., 1982.

7. J.M. Delosme and M. Morf, "Mixed and Minimal Representations for Toeplitz and Related Systems," in *Proc. 14th Asilomar Conf. Circuits, Systems & Computers* (Pacific Grove, Calif., Nov. 1980), pp. 19-24.

8. ——, "A Tree Classification of Algorithms for Toeplitz and Related Equations Including Generalized Levinson and Doubling Type Algorithms," in *Proc. 19th IEEE Conf. Decision & Control* (Dec. 1980), pp. 42-66.

9. B. Friedlander, T. Kailath, M. Morf, and L. Ljung, "Extended Levinson and Chandrasekhar Equations for General Discrete-Time Linear Estimation Problems," *IEEE Trans. Automat. Cont.* **AC-23** (4), 653-659, Aug. 1978.

10. B. Friedlander, M. Morf, T. Kailath, and L. Ljung, "New Inversion Formulas for Matrices Classified in Terms of Their Distance from Toeplitz Matrices," *Linear Algebra and Its Applications* **27**, 31-60, 1979.

11. T. Kailath, L. Ljung, and M. Morf, "Generalized Krein-Levinson Equations for Efficient Calculation of Fredholm Resolvents of Nondisplacement Kernels," in *Topics in Functional Analysis* (essays in honor of M.G. Krein, Advances in Mathematics Supplementary Studies, Vol. 3) (New York: Academic, 1978).

12. T. Kailath, A. Vieira, and M. Morf, "Inverses of Toeplitz Operators, Innovations, and Orthogonal Polynomials," *SIAM Rev.* **20** (1), 106-119, 1978.

13. L. Ljung, M. Morf, and D. Falconer, "Fast Calculations of Gain Matrices for Recursive Estimation Schemes," *Int. J. Cont.* **27** (1), 1-19, 1978.

14. M. Morf, "Fast Algorithms for Multivariable Systems," Ph.D. dissertation, Dept. Elec. Eng., Stanford University, Stanford, Calif., 1974.

15. I.S. Reed, J.D. Mallet, and L.E. Brennan, "Rapid Convergence Rate in Adaptive Arrays," *IEEE Trans.* **AES-10**, 853-863, Nov. 1974.

16. S.P. Applebaum, "Adaptive Arrays," *IEEE Trans.* **AP-24** (5), 585-598, Sept. 1976.

17. L.E. Brennan, J.D. Mallett, and I.S. Reed, "Adaptive Arrays in Airborne MTI Radar," *IEEE Trans.* **AP-24**, (5), 607-614, Sept. 1976.

18. G.A. Andrews, "Optimal Radar Doppler Processors," NRL Report 7727, May 1974.